Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Spring Term 2018

ADVANCED COMPUTER NETWORKS
Assignment 8: Introduction to RDMA Programming

Assigned on: **08 May 2018**
Due by: **26 May 2018, 23:59**

1 Introduction

The goal of this project is to give an introduction to RDMA programming [5] and related software [6, 2]. This exercise assumes basic knowledge of Linux and Java.

You will use DiSNI [2] for building RDMA applications. It is a Java library for direct storage and networking access from userpace. You will use SoftiWARP [6] which emulates RDMA hardware and implements the iWARP protocol suite in software. Both DiSNI and SoftiWARP are developed and maintained by the High-performance IO Research Group at IBM Research Lab, Zurich.

Below are some important points:

- This is a group project with the maximum group size of 2. You are free to use Slack to find your group member. It is **mandatory for each member** to upload solution to the Git repo.
- The assignment is due on 26 May 2018. We will have 2 clinic sessions on 17 May and 24 May. During the first session, we will briefly introduce you to the assignment. **You are highly encouraged to join the sessions and to get your doubts clarified.**
- **The TAs are not responsible to address your individual queries on slack** and tagging them on posts is not recommended. In the context of this assignment, Slack is meant for the students to discuss setup and configuration issues. Nevertheless, TAs will monitor the channels and might step in at any point to address critical concerns.
- Read section 4 and 5 before starting with the implementation.

2 Let's start

Download the Ubuntu 16.04 LTS VM image available in the course website for this assignment. We have installed all the software required to run the RDMA applications including SoftiWARP and DiSNI (*/home/student/Development*). You are free to setup your own environment and install these software along with their associated dependencies. A good starting point can be [1].

Also download the static content zip archive available in the course website which contains an HTML file and an image needed to complete the assignment [7].

2.1 Check the VM

The username for the VM is `student` (password: `student`) and you should have `sudo` access with this user.

Below are some commands which you should run after you start the VM. If something does not seem to work fine, check the appendix for possible fixes.

- `ibv_devices` : Lists RDMA devices available for use from userspace. You should see two devices `siw_eth` and `siw_lo`
- `ibv_devinfo` : Prints information about RDMA devices available for use from userspace.

If you do not see any devices, try to run `sudo modprobe siw`. That should load the SoftiWARP kernel module and activate your devices. In the VM, SoftiWARP is precompiled for the kernel `4.8.0-36-generic`. That should be your default kernel.

2.2 Familiarize yourself with DiSNI

DiSNI has a few great examples [3] which can immediately get you started. Try to run some of these and have a look at the code in order to understand the APIs. A good start can be the SendRecvClient and SendRecvServer.

3 The real assignment

In this assignment you need to build an HTTP Server which serves static content and an HTTP client-side proxy. For convenience, both the proxy and the server can be implemented in the same VM. The assignment is sub-divided into segments. You should try to complete these segments one after another in order.

3.1 A server which just returns the HTTP response code

In this part, build a client-side proxy which intercepts HTTP requests from any browser of your choice. The proxy is pretty naive and it sends a 404 HTTP Response code back to the browser unless the browser sends a `GET /` HTTP request to the web server at `www.rdmawebpage.com`. In this particular case, the proxy forwards the request to the server. The server replies back with a 200 OK HTTP Response code which is forwarded by the proxy to the browser. The proxy and the server should be 2 different applications listening at different ports. This part of the assignment must be implemented using SEND/RECV. If the communication between the proxy and the server fails, the proxy should reply with HTTP 504 (Gateway Time-out).

3.2 A server which returns the HTML content

The server returns the HTML content size and other necessary parameters to the client proxy along with the HTTP response code. The client proxy should fetch the HTML content (file provided in the static content zip archive) from the server's memory using an RDMA READ operation. After fetching, the content should be forwarded to the browser for rendering along with the HTTP response code.

3.3 A server which serves HTML content + image

This is a simple extension of the previous step. The browser starts parsing the HTML content and encounters a URL to an image. The browser issues request for the image which is intercepted by the client proxy. The client-side proxy fetches the image from the server in the way it fetched HTML content and forwards it to the browser along with the received HTTP response code. The browser loads the HTML content along with the image. The web page as rendered in the browser at the end of thios step should look similar to Figure 1.

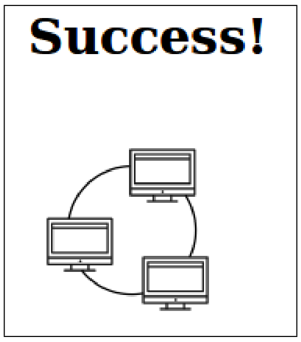

Figure 1: HTML content + image

3.4 Handling multiple clients and requests

Modify your server to handle multiple requests from multiple client-side proxies. The requests do not necessarily need to be parallel.

4 Choosing the EndpointGroup

EndpointGroups are containers and factories for RDMA connections (RdmaEndpoint). There are two types of groups available in the RDMA API, and which type works best depends on the application. The RdmaActiveEndpointGroup actively processes network events caused by RDMA messages being transmitted or received. Events are signaled by calling dispatchCqEvent() which can be overriden by the custom endpoint of the application. The RdmaPassiveEndpointGroup provides a polling interface that allows the application to directly reap completion events from the network queue (completion queue). As such, the passive mode has typically lower latency but may suffer from contention in case of large numbers of threads operating on the same connection. The active mode, on the other hand, is more robust under large numbers of threads, but has higher latencies. For this assignment we suggest you to use RdmaActiveEndpointGroup at both the server and the client proxy.

5 Useful Hints

At the server, allocate an endpoint group and initialize it with the factory, create a server endpoint, bind it and accept connections. At the client, also create a custom endpoint and factory (not shown) and connect your endpoint to the server. Once an endpoint is connected, RDMA data operations can be issued. For more details, kindly refer to the section `Programming RDMA using`

`DiSNI` in the DiSNI github page [2]. We also suggest you to go through the example applications [3] in order to understand the APIs.

Other useful tips:

- We have installed the necessary software including SoftiWARP and DiSNI in the VM. So you should not need to run commands like `./autoconfigure.sh` or `sudo ./configure` at all. To run the examples you need to set the `LD_LIBRARY_PATH` as follows: `export LD_LIBRARY_PATH=/usr/local/lib`
- Command to run the ReadServer example: `java -Djava.library.path=/usr/local/lib com.ibm.disni.examples.ReadServer -a <IP>`
- While running your code, if you get an exception like `java.io.IOException: setting up protection domain, no context found`, you have to bind the server to the actual IP address of one of your interfaces except loopback and not `127.0.0.1`.
- If you suddenly experience `java.io.IOException: No RDMA device configured!`, executing command `ibv_devices` no longer shows any device and `sudo modprobe siw` throws `FATAL: Module siw not found in directory ...`, the kernel version might have changed. Downgrade it to 4.8.0-36-generic with grub-customizer and things should work.

6 Git Instructions

- You are going to use Git to handin your assignments for this course. Make sure you have it installed on your computer.
- Clone your private repository by typing:

  ```
  git clone git@gitlab.inf.ethz.ch:COURSE-ACN-2018/NETHZ_USERNAME.git
  ```

  or

  ```
  git clone https://gitlab.inf.ethz.ch/COURSE-ACN-2018/NETHZ_USERNAME.git
  ```

  You can access the repository online

  `https://gitlab.inf.ethz.ch/COURSE-ACN-2018/NETHZ_USERNAME`
- For more information about how to use Git, look at [4]

7 Hand-In Instructions

- You should use following directory structure to submit your solution:

  ```
  assignment8
  |-- RDMAServer
  |-- RDMAClientProxy
  `-- README.txt
  ```

- The server code and the client proxy code should be uploaded to the respective folders.

- Your code should be well-documented.
- Write a one-page report (README.txt) to let us know how your application works or to explain if we need to do something extra in order to run your solution.

References

[1] Detailed blog about Installation. `http://www.reflectionsofthevoid.com/2011/03/how-to-install-soft-iwarp-on-ubuntu.html`.

[2] DiSNI: Direct Storage and Networking Interface. `https://github.com/zrlio/disni`.

[3] DiSNI: Examples. `https://github.com/zrlio/disni/tree/master/src/test/java/com/ibm/disni/examples`.

[4] Git tutorial. `https://git-scm.com/docs/gittutorial`.

[5] RDMA Consortium. `http://www.rdmaconsortium.org/`.

[6] SoftiWARP: Software iWARP kernel driver and user library for Linux. `https://github.com/zrlio/softiwarp`.

[7] Static content. `https://www.systems.ethz.ch/sites/default/files/file/COURSES/2017_SPRING_COURSES/ACN/a_8_static_content.zip`.